Peer to Peer Networks

Prepared by T. Le-Duc and H. Choo
Presentation Outline

- 2.1 Introduction
- 2.2 Client-Server Paradigm
- 2.3 Peer-To-Peer Paradigm
2.1 Introduction

- The whole Internet was designed and developed to provide services at the application layer
- The application layer provides services to the Internet user
- Communication is provided using a logical connection
  - Two applications assume that there is an imaginary direct connection through which they can send and receive messages
2.1 Introduction

Scenario

- A scientist at Sky Research needs to order a book from an online bookseller, Scientific Books

[Figure 2.1 Logical connection at the application layer]
2.1 Introduction

Providing Services

- Applications and services are added to the Internet constantly

- **Standard** Application-Layer protocols
  - Protocols that have been standardized and documented by the Internet authority
  - E.g. HTTP, FTP, SMTP, POP3, Telnet, …

- **Nonstandard** Application-Layer protocols
  - Protocols do not need the approval of the Internet authority
  - E.g. customized protocol developed by a company to communicate with all its offices
2.1 Introduction
Application-Layer Paradigms: Client-Server paradigm (1/2)

- **Server**
  - Always-on host
  - Provides requested service to client
  - Permanent IP address

- **Client**
  - Initiates contact with server
  - Typically requests service to server
  - Do not communicate directly with each other

- WWW, FTP, Remote login (Telnet, SSH), e-mail, …
2.1 Introduction

Application-Layer Paradigms: Client-Server paradigm (2/2)

[Figure 2.2 Example of a client-server paradigm]
2.1 Introduction

Application-Layer Paradigms: Peer-to-Peer Paradigm (1/2)

- Peer-to-peer (P2P) paradigm
  - No always-on server
  - Arbitrary end systems directly communicate
  - Peers are intermittently connected and change their IP address
  - Highly scalability, increase service capability
  - Highly distributed and decentralized nature
  - Difficult to manage
2.1 Introduction

Application-Layer Paradigms: Peer-to-Peer Paradigm (2/2)

[Figure 2.3 Example of a peer-to-peer paradigm]
2.1 Introduction
Application-Layer Paradigms: Mixed Paradigm

- Mixed paradigm (Client-Server paradigm + P2P)
  - Client-Server paradigm: a client gets the information from a server
  - P2P: the client can share the information with other clients
  - E.g. Nate-on messenger, Yahoo messenger

[Example of Mixed paradigm]
2.2 Client-Server Paradigm

Application Programming Interface

- Communication at the application layer is between two applications called processes: a client and a server
  - A client is a running program that initializes the communication by sending a request
  - A server is another application program that waits for a request from a client

- Application Programming Interface (API)
  - A computer language has a set of instructions for mathematical operations, a set of instructions for string manipulation, a set of instructions for input/output access, and so on
  - A set of instructions to tell the lowest four layers of the TCP/IP suite to open connection, send and receive data, and close the connection is referred as an API for networking applications
2.2 Client-Server Paradigm
Application Programming Interface: Sockets (1/4)

- Socket interface started in the early 1980s at UC Berkeley as part of a UNIX environment
- The socket interface is a set of instructions that provide communication between the application layer and the operating system
- Socket
  - Although a socket is supposed to behave like a terminal or a file, it is not a physical entity like them
  - It is a data structure that is created and used by the application program
2.2 Client-Server Paradigm
Application Programming Interface: Sockets (2/4)

[Figure 2.4 Position of the socket interface]
2.2 Client-Server Paradigm

Application Programming Interface: Sockets (3/4)

Application program

- Read
- Write

Keyboard (source)
Monitor (sink)
File (sink and source)
Socket (sink and source)

[Figure 2.5 Sockets used the same way as other sources and sinks]
2.2 Client-Server Paradigm

Application Programming Interface: Sockets (4/4)

- Sockets can be used for the communication between a client process and server process.
- The client thinks that the socket is the entity that receives the request and gives the response.
- The server thinks that the socket is the one that has a request and needs the response.

[Figure 2.6 Use of socket in process-to-process communication]
2.2 Client-Server Paradigm

Using Services of the Transport Layer

- A pair of processes provide services to the Internet users, human or programs

- A pair of processes, however, need to use the services provided by the transport layer for communication because there is no physical communication at the application layer

- There are 3 common transport layer protocols
  - UDP
  - TCP
  - SCTP
2.2 Client-Server Paradigm

Using Services of the Transport Layer: UDP Protocol

- UDP provides **connectionless, unreliable, datagram** service
  - Connectionless service means that there is no logical connection between the two ends exchanging messages

- Each message is an **independent entity** encapsulated in a packet called a datagram

- UDP does not see any relation (connection) between consequent datagrams coming from the same source and going to the same destination

- UDP is suitable for applications
  - Small messages usage
  - Simplicity and speed are more important than reliability
  - E.g.: management and multimedia applications
2.2 Client-Server Paradigm
Using Services of the Transport Layer: TCP Protocol

- TCP provides connection-oriented, reliable, byte-stream service
- TCP requires that two ends first create a logical connection between themselves by exchanging some connection-establishment packets
  - Handshaking process: establishes some parameters between the two ends including the size of the data packets to be exchanged
- Most of the standard applications that need to send long messages and require reliability may benefit from the service of the TCP
  - E.g.: File transfer applications
2.3 Standard Client-Server Application

World Wide Web and HTTP: Introduction (1/2)

■ Video Content

► The World Wide Web (WWW) is used every day by millions of people for everything from checking the weather to sharing videos

► This interconnected information system (WWW) is described as a virtual city that everyone owns and explains how it is organized in a way that mimics our brain's natural way of thinking

► Link: https://www.youtube.com/watch?v=J8hzJxb0rpc

The World Wide Web is NOT the Internet
2.2 Client-Server Paradigm

World Wide Web and HTTP: Introduction (2/2)
2.3 P2P Paradigm

P2P Applications

- Traditional P2P applications: for file sharing
  - BitTorrent, Emule etc.
  - Quickly find any type of file for free

- IP Telephony
  - Skype: Using P2P overlay network
  - Using Skype clients to place voice calls and send text messages
2.3 P2P Paradigm

P2P Applications

- Block-chain based P2P trading/transaction application, for crypto-currencies such as Bitcoin, Ethereum, Litecoin

- P2P cloud storage: promisingly strong security and reliability
  - Data is stored in different locations, broken into blocks, further shred into fragments
  - Nearly impossible for anyone except data owner to gain access
  - Example: Cloudplan, CrashPlan, Storj
A peer-to-peer network is the one in which each computer can act as a client or server for the other computers.

When a peer in the network has a file (e.g., an audio or video file) to share, it makes it available to the rest of the peers.

P2P networks can be divided into two categories: centralized and decentralized networks.
2.3 P2P Paradigm

P2P Networks: Centralized Networks

- In a centralized P2P network,
  - The directory system – listing of the peers and what they offer – uses the client-server paradigm
  - Only the storing and downloading of the file are done using the peer-to-peer paradigm
A decentralized P2P network does not depend on a centralized directory system.

In this model, peers arrange themselves into an overlay network (logical network made on top of the physical one).

A decentralized P2P network is classified as either unstructured or structured networks.
Unstructured Networks

- The nodes are linked randomly
- A search in an unstructured P2P is not very efficient because a query to find a file must be flooded through the network
- This network type produces significant traffic and still the query may not be resolved
- Two example of this type of network are Gnutella and Freenet (for file sharing)
Structured Networks

- A structured network uses a predefined set of rules to link nodes so that a query can be effectively and efficiently resolved

- The most common technique used for this purpose is the Distributed Hash Table (DHT)

- One popular P2P file sharing protocol that uses the DHT is BitTorrent
2.3 P2P Paradigm

Distributed Hash Table (1/5)

- A Distributed Hash Table (DHT) distributes data among a set of nodes according to some predefined rules.
- In a DHT-based network, each data item and the peer is mapped to a point in a large address space of size $2^m$.
- Most of the DHT implementations use $m=160$.

![Figure 2.47 Address space]

Note:
1. Space range is 0 to $2^m - 1$.
2. Calculation is done modulo $2^m$. 
2.3 P2P Paradigm

Distributed Hash Table (2/5)

- Hashing Peer Identifier
  - The first step in creating the DHT system is to place all peers on the address space ring.
  - This is normally done by using a hash function that hashes the peer identifier, normally its IP address, to an m-bit integer, called a node ID.
    \[
    \text{node ID} = \text{hash (Peer IP address)}
    \]

- Hashing Object Identifier
  - The name of the object (e.g., a file) to be shared is also hashed to an m-bit integer in the same address space.
  - The result in DHT notation is called a key.
    \[
    \text{key} = \text{hash (Object name)}
    \]
2.3 P2P Paradigm
Distributed Hash Table (3/5)

- Storing the Object
  - Assume that $m=5$ and several peers have already joined the group
  - The node $N5$ has a file named *Liberty* that wants to share with its peers
    - Reference to this file is stored at the closest node to the hashed key of the file

![Legend](image)

[Distributed Hash Table](#)

![Figure 2.48 Example 2.15](image)
2.3 P2P Paradigm

Distributed Hash Table (4/5)

■ Routing
  ▶ Each node needs to have a partial knowledge about the ring to route a query to a particular node which is responsible for storing the reference to an object

■ Arrival and Departure of Nodes
  ▶ When a computer launches the DHT software, it joins the network; when it is turned off or the peer closes the software, it leaves the network
  ▶ The arrival or departure of the nodes need to be handled by a clear and efficient strategy
There are several protocols that implement DHT systems.

Three of these protocols, **Chord**, **Pastry**, and **Kademlia**, will be discussed for some reasons:

- **Chord protocol**: due to its simplicity and elegant approach to routing queries.
- **Pastry protocol**: different approach than Chord, but close to Kademlia protocol in routing strategy.
- **Kademlia protocol**: used in the most popular file-sharing network, BitTorrent.
2.4 P2P Paradigm

BitTorrent: Introduction (1/2)

- Video Content
  - What BitTorrent is and Why BitTorrent?
  - An explanation of sharing huge contents on the Internet
  - How BitTorrent works?
  - Link: https://www.youtube.com/watch?v=6PWUCFmOQwQ
2.3 P2P Paradigm

BitTorrent: Introduction (2/2)
Data items and nodes in Chord create an identifier space of size $2^m$ points distributed in a circle in the clockwise direction.

- We refer to the identifier of a data item as $k$ (for `key`) and the identifier of a peer as $N$ (for `node`).

Arithmetic in the space is done modulo $2^m$, which means that the identifiers are wrapped from $2^m - 1$ back to $0$.

The closest peer with $N \geq k$ (key), called the `successor of k`, hosts the value $(k, v)$, where $v$ is information about the peer server that has the object.

- The peer that stores the data item and the peer that holds the pair $(k, v)$ are not necessarily the same.
2.3 P2P Paradigm
Chord: Finger Table (1/2)

- A node in the Chord algorithm should be able to resolve a query: given a key, the node should be able to find the node identifier responsible for that key or forward the query to another node.
- Each node needs to have a routing table, called a finger table by Chord.
- The finger table of a node contains entries for \( m \) Successors:
  - \( i^{th} \) entry contains a reference to the first node that succeeds this node by at least \( 2^{(i-1)} \) on the identifier circle.
- Each node also stores one Predecessor.

<table>
<thead>
<tr>
<th>( i )</th>
<th>Target Key</th>
<th>Successor of Target Key</th>
<th>Information about Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( N + 1 )</td>
<td>Successor of(N + 1)</td>
<td>IP address and port of successor</td>
</tr>
<tr>
<td>2</td>
<td>( N + 2 )</td>
<td>Successor of(N + 2)</td>
<td>IP address and port of successor</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( m )</td>
<td>( N + 2^{m-1} )</td>
<td>Successor of(N + 2(m-1))</td>
<td>IP address and port of successor</td>
</tr>
</tbody>
</table>

[Table 2.14 Finger table]
2.3 P2P Paradigm
Chord: Finger Table (2/2)

The successor of \( k \) hosts the information about the peer server that has the object.

The finger table of \( N5, N10 \)

---

**Table 2.14 Finger table**

<table>
<thead>
<tr>
<th>( i )</th>
<th>Target Key</th>
<th>Successor of Target Key</th>
<th>Information about Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( N + 1 )</td>
<td>Successor of ((N + 1))</td>
<td>IP address and port of successor</td>
</tr>
<tr>
<td>2</td>
<td>( N + 2 )</td>
<td>Successor of ((N + 2))</td>
<td>IP address and port of successor</td>
</tr>
<tr>
<td>( m )</td>
<td>( N + 2^{m-1} )</td>
<td>Successor of ((N + 2^{m-1}))</td>
<td>IP address and port of successor</td>
</tr>
</tbody>
</table>

---

**Figure 2.49 An example of a ring in Chord**

[Diagram showing a ring with nodes and successor links, including tables for target keys and successors.]
2.3 P2P Paradigm
Chord: Interface (1/7)

- Lookup
  - The mostly used operation in Chord

```plaintext
Lookup (key)
{
    if (node is responsible for the key)
        return (node's ID)
    else
        return find_successor (key)
}

find_predecessor (id)
{
    x = N
    // N is the current node
    while (id \notin (x, x.finger[1])
    {
        x = x.find_closest_predecessor (id)
    }
    return x
}

find_closest_predecessor (id)
{
    for (i = m downto 1)
    {
        if (finger [i] \in (N, id))
            return (finger [i])
    }
    return N
}
```
Assume node N5 needs to find the responsible node for key k14.

[Figure 2.50 Example 2.16]

[Figure 2.49 An example of a ring in Chord]
2.3 P2P Paradigm
Chord: Interface (3/7)

- **Stabilize**
  - Each node in the ring periodically uses this operation to validate its information about its successor and let the successor validate its information about its predecessor.

```java
Stabilize()
{
    P = finger[1].Pre //Ask the successor to return its predecessor
    if (P ∈ (N, finger[1])) finger[1] = P // P is the possible successor of N
    finger[1].notify (N) // Notify P to change its predecessor
}

Notify (x)
{
    if (Pre = null or x ∈ (Pre, N)) Pre = x
}
```
2.3 P2P Paradigm
Chord: Interface (4/7)

- **Fix_Finger**
  - Each node in the ring must periodically call this operation to maintain its finger table update
  - One finger is chosen randomly to be updated in each call

```java
Fix_Finger()
{
    Generate (i ∈ (1, m)) // Randomly generate i such as 1 < i ≤ m
    finger[i] = find_successor(N + 2^{i-1}) // Find value of finger[i]
}
```
2.3 P2P Paradigm

Chord: Interface (5/7)

- **Join**
  - When a peer joins the ring, it uses the join operation and known ID of another peer to find its successor and set its predecessor to null.
  - Stabilize function is called then to validate its successor.
  - Move-key function is called to ask the successor to transfer the keys that this node is responsible for.

```plaintext
Join (x)
{
    // N is the current node
    Initialize (x)
    finger[1].Move_Keys (N)
}

Initialize (x)
{
    Pre = null
    if (x = null) finger[1] = N
    else finger[1] = x. Find_Successor (N)
}

Move_Keys (x)
{
    for (each key k)
    {
        if (x ∈ [k, N)) move (k to node x)
    }
}
Assume that node \textbf{N17} joins the ring with the help of \textbf{N5}

1. N17 set its predecessor to null and its successor to N20 by \textit{Initialize}(5)
2. N17 then asks N20 to send k14 and k16 to N17 by \textit{Move_Keys}(17)
3. N17 validates its own successor by \textit{Stabilize} and asks N20 to change its predecessor to N17
4. The predecessor of N17 is updated to N12 when N12 uses \textit{Stabilize}
5. The finger table of nodes N17, N10, N5, and N12 is changed by \textit{Fix_Finger}

[Figure 2.51: Example 2.17]
2.3 P2P Paradigm
Chord: Interface (7/7)

- **Leave or Fail**
  - The ring must be stabilized if a peer leaves or fails

- Assume that node **N10** leaves the ring

1. Node N5 detects N10’s departure, then changes its successor to N12 (the second in the list of successors)

2. Node N5 launches the *Stabilize* function and asks N12 to change its predecessor to N5

3. Hopefully, k7 and k9, which were under the responsibility of N10, have been duplicated in N12 before the departure of N10

4. After a few calls of *Fix_Finger*, nodes N5 and N25 update their finger tables

[Figure 2.52: Example 2.18]
2.3 P2P Paradigm

Kademlia

- Kademlia, like Pastry, routes messages based on the **distance between nodes**

- The distance between the two identifiers (nodes or keys) is measured as the bitwise exclusive-or (XOR) between them

- In other words, if \( x \) and \( y \) are two identifiers:
  \[
  \text{distance} \ (x, \ y) = x \oplus y
  \]

- For example, \( 12 \ (1100)_{2} \oplus 11 \ (1011)_{2} = 7 \ (0111)_{2} \)
2.3 P2P Paradigm
Kademlia: Identifier Space

- In Kademlia, nodes and data items are \( m \)-bit identifiers that create an identifier space of \( 2^m \) points distributed on the leaves of a binary tree
  - Assume that \( m=4 \), we have 16 (\( 2^4 \)) identifiers in the identifier space
  - The key \( k_3 \) is stored in \( N_3 \) because \( 3 \oplus 3 = 0 \)
  - Although the key \( k_7 \) looks numerically equidistant from \( N_6 \) and \( N_8 \), it is stored only in \( N_6 \) because \( 6 \oplus 7 = 1 \) but \( 6 \oplus 8 = 14 \)
2.3 P2P Paradigm
Kademia: Routing Table (1/3)

- Each node has \( m \) subtrees corresponding to \( m \) rows in the routing table, but only one column
- Subtree \( i \) includes nodes that share \( i \) leftmost bit (common prefix) with the corresponding node
  - Assume that each row holds the identifier of one of the nodes in the corresponding subtree

<table>
<thead>
<tr>
<th>Common prefix length</th>
<th>Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Closest node(s) in subtree with common prefix of length 0</td>
</tr>
<tr>
<td>1</td>
<td>Closest node(s) in subtree with common prefix of length 1</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( m-1 )</td>
<td>Closest node(s) in subtree with common prefix of length ( m-1 )</td>
</tr>
</tbody>
</table>

[Table 2.21: Routing table for a node in Kademia]
2.3 P2P Paradigm
Kademlia: Routing Table (2/3)

- With \( m=4 \), each node has four subtrees corresponding to 4 rows in the routing table.
- In case of **row 0** (common prefix \( p=0 \)) in \( N6 \), N15 inserted in row 0 because N15 is the closest to N6 (\( N6 \oplus N15 = 9 \), \( N6 \oplus N8 = 14 \), \( N6 \oplus N11 = 13 \)).

![Routing Table Diagram](image_url)

**Subtrees for node N6**

[Figure 2.56: Example 2.24]
Assume that node **N0** receives a lookup message to find the node responsible for **k12**

The length of the common prefix between **N0**, **k12** is 0; so node N0 sends the message to the node in row 0 of its routing table, node **N8**

The length of the common prefix between **N8** and **k12** is 1; so node N8 sends the message to the node in row 1, node **N15**, which is responsible for **k12**
For more efficiency, Kademlia requires that each row keeps at least up to $k$ (around 20) nodes from the corresponding subtree.

- Each row in the routing table is referred to as a $k$-bucket.
- Redundancy is for tolerance of the leaving or failure of nodes.

**Parallel Query**

- Kademlia allows sending $\alpha$ parallel queries to $\alpha$ nodes at the top of the $k$-bucket for reducing the delay in case some node fails.

**Concurrent Updating**

- Whenever a node receives a query or a response, it updates its $k$-bucket.
- If multiple queries to a node receive no response, the sender of the query removes the destination node from the corresponding $k$-bucket.
2.3 P2P Paradigm
Kademlia: Join / Leave or Fail

- Node joins the network
  - It needs to know at least one other node
  - It sends its identifier to the known node as if it is a key to be found; and the response allows it to create its k-bucket

- Node leaves or fails
  - When a node leaves the network or fails, other nodes update their k-buckets using the aforementioned concurrent process
2.3 P2P Paradigm
A Popular P2P Network: BitTorrent (1/3)

- **BitTorrent** is a P2P protocol for sharing a large file among a set of peers
- File sharing is done in a collaborating process called a **torrent**
- Each peer participating in a torrent downloads chunks of the large file from another peer (that has them) and uploads chunks of that file to other peers

- Set of all peers in a torrent is referred to as a **swarm**
  - A peer that has the complete content file is called a **seed**
  - A peer that has only part of the file and wants to download the rest is called a **leech**

- BitTorrent has gone through several versions and implementations
  - Original one using a central node, called **tracker**
  - New versions without tracker
2.3 P2P Paradigm

A Popular P2P Network: BitTorrent (2/3)

- BitTorrent with a Tracker
  - A Tracker is a server that assists in the communication between peers using BitTorrent protocol
  - To download a data file, a peer
    - Must have a torrent file, from BitTorrent server, containing information of pieces in the content file and address of the tracker handling that specific torrent
    - Contacts the Tracker to get information of seeds and leeches in the torrent
    - Starts downloading data chunks from seeds and leeches

![Figure 2.57 Example of a torrent with Tracker]

Note: Peers 2 and 4 are seeds; others are leeches.
2.3 P2P Paradigm
A Popular P2P Network: BitTorrent (3/3)

- Trackerless BitTorrent
  - The job of tracking is distributed among some nodes in the network
    - Some nodes play the role of trackers
  - Kademlia DHT can be used for the implementation
    - Key: metadata file that defines the torrent
    - Value: list of peers in the torrent (seeds and leeches)
    - The Kademlia protocol finds the node responsible for a given key, and then returns the list of peers in the corresponding torrent
      - A joining peer can use the BitTorrent protocol to share/download the content file with/from peers in the list
Practice Problems

1. In a DHT-based network, what is the size of peer-identifier space? And what is the range of values in this space?

2. In Kademlia, assume $m=4$ and active nodes are N4, N7, and N12. Where is the key k3 stored in this system?